Sissejuhatus infotehnoloogiasse

Tarkvara arhitektuur

Sisukord

- Arhitektuurinduse ja arendustöö põhimõtteid
- Komponendid: terviksüsteemid, serverid, frameworkid, teegid
- Arhitektuurinäiteid
- Programmeerimiskeelte tüübid eri vaadetest
- Domineerivad programmeerimiskeeled
- Arendusvahendid
- Tarkvara & tootmine: ajaloolised faasid
- Võrgunduse areng
- Süsteemide integratsioon
- Vabatarkvara

Tarkvara arhitektuur

- Arhitektuuri all mõeldakse IT-s:
  - mingi süsteemi tehnoloogilisi põhimõtteid
  - millisteks suurteks osadeks süsteem jaotub
  - kuidas osad omavahel suhtlevad
  - milliseid suuri valmistükke süsteem kasutab

- Tarkvaras on sellisteks küsimusteks näiteks:
  - Mis opsüsteemi alla rakendus teha
  - Kas hoida infot lihtsalt failides või andmebaasis
  - Millist veebiserverit ja kuidas täpselt kasutada
  - Kas pöörduda andmebaasi poole otse või csv või XML või json vahekihi abil
  - Mis keeles/keeltes rakendus teha
  - Millist kasutajaliidese teeki kasutada
  - ... jne ....

Arendaja analoog on arhitekt/ehitusinsener

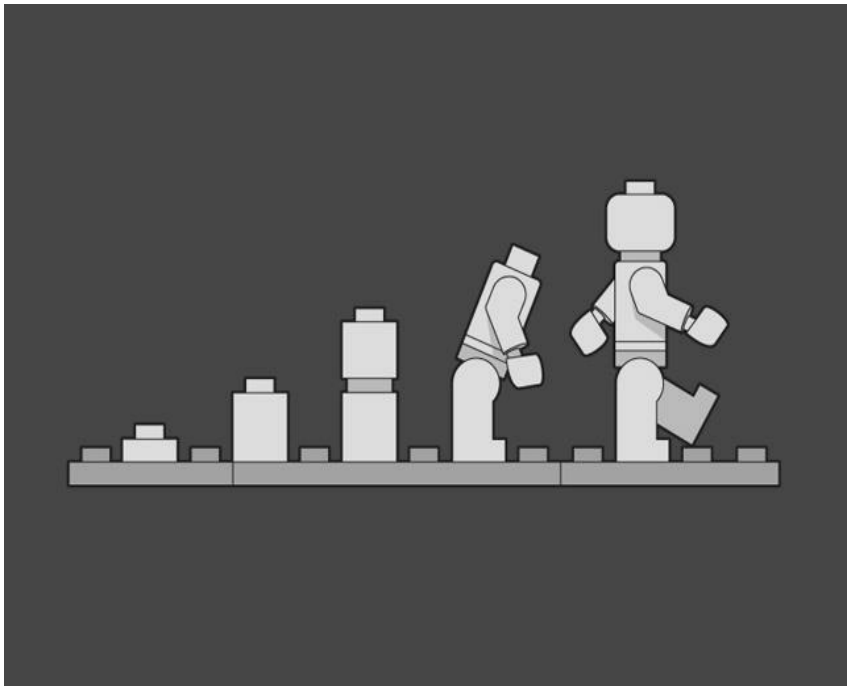
Tarkvara lihtne analoogia ehitusega ei ole eriti pädev:

- Tarkvara on kõige keerulisem asi maailmas: tarkvarasüsteemidel on kõige rohkem osi
- Arendaja kirjutatud programm ongi tegelikult ehitusjoonis, mitte ehitus: ehitus (töötav tarkvara) tehakse kompilaatorite, linkurite jms abil programmist automaatselt.
- Programmeerija “analoog” on arhitekt/ehitusinsener.
- Tarkvara arenduses ei ole nö lihtsaid “ehitusmehi” vaja.
- Täiendava tööjõu lisamine projekti käigus teeb projekti täitmise reeglina veel aeglasemaks.

Tarkvara ei ole lego

Tarkvara ei panda kokku “nagu lego blokkidest”:

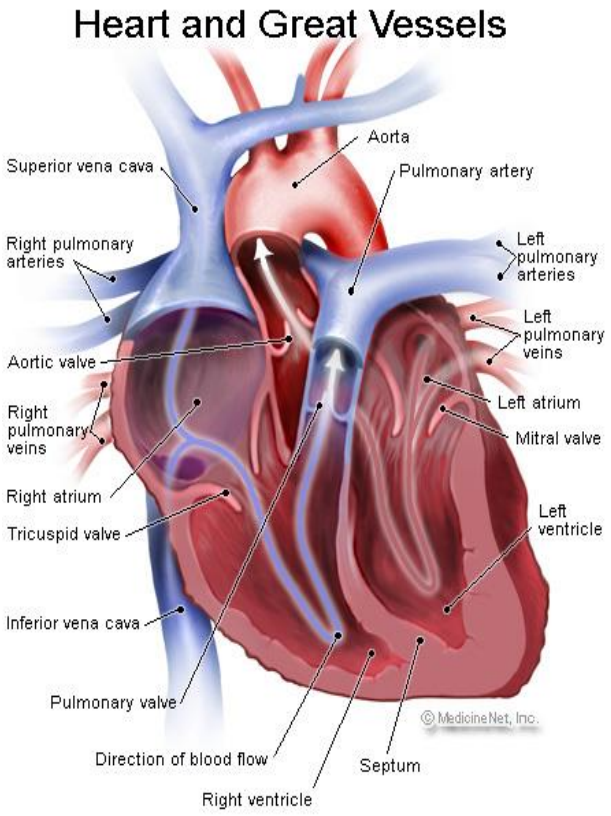
- Lego blokke saab ühendada väga lihtsalt, ja neil ei ole sisemist keerulist ehitust



Tarkvara on sarnane bioloogiaga

Tarkvara ei panda kokku “nagu lego blokkidest”:
ta sarnaneb enim **bioloogiaga**

- “Tarkvarablokid” on väga keerulised süsteemid, ja nende ühendamine tähendab nö tavapärast programmeerimist
- Tarkvara areneb olemasolevast tarkvarast arendajate abil evolutsiooni teel, sarnaselt bioloogiaga



Arhitektuuri eesmärgid: lihtsus mitmekesisuses

Arendamise ja haldamise lihtsus on kõige tähtsam.

- Valiku juures on otsustav see, mida arendajad/haldajad kõige paremini oskavad kasutada.
- Lisatükkide kasutamist ilma selge vajaduseta tuleks vältida.
- Abstraktsioonid tilguvad läbi.
- Ei ole olemas iga juhu jaoks sobilikke tehnoloogiaid/komponente

No silver bullet!

Tarkvarasüsteemide komponendid

Tarkvarasüsteemid ehitatakse reeglina mitmesuguste komponentide kokkupaneku, s.t. kokkuprogrammeerimise teel, või teisiti öeldes: komponente kasutades.

Neid komponente võib klassifitseerida - näiteks - järgmisel viisil:

- Terviklikud lõppkasutaja-rakendusprogrammid
- Suured “valmiskomponendid“, näiteks andmebaasimootorid
- Raamistikud ehk frameworks
- Teegid ehk libraries

Tarkvarasüsteemide komponendid

Terviklikud lõppkasutaja-rakendusprogrammid,

Neid saab tihti juhtida ja mõne teise tarkvarapaketiga programmiliselt siduda, a la:

- Tekstitöötlus, näiteks Word või LibreOffice
- Tabelarvutus, näiteks Excel
- Raamatupidamise tarkvara
- Arendustarkvara, näiteks Eclipse või vscode
- Veebibrauser

Tarkvarasüsteemide komponendid

Suured valmiskomponendid

Andmebaasiserverid, www-serverid, mailiserverid, graafikaserverid nagu X11, operatsioonisüsteem ise jne

Need programmid on tehtud eeskätt selleks, et lõppkasutaja jaoks loodud rakenduste tegemist hõlbustada. Näitena andmebaasi väljakutsumine Pythonist:

```
.....
cursor=con.cursor()
sqlstr= """  select clientid, clientname
             from clientbase
             where clientname like '%Jaan%' """
cur.execute(sqlstr)
results=cur.fetchall()
for i in results
    print "id: ",i[0], " name: ",i[1]
.....
```

Tarkvarasüsteemide komponendid

Raamistikud ehk frameworks

Edasiarendamiseks ja ümbertegemiseks mõeldud terviklikud näite-rakendused, levinud eeskätt „tüüpiliste“ andmebaasi-kesksete veebirakenduste jaoks. Igast hästi läbimõeldud rakendusest võib saada selline näiterakendus. Populaarseid raamistikke:

- Java Spring
- Ruby on Rails
- PHP Zend framework
- Python Django
- Javascript Angular
- Microsoft .NET

Raamistik ei lase arendajal vabalt valida, kuidas süsteem peaks töötama, vaid pigem suunab teda täiendama etteantud näite-süsteemi, mille raamistiku arendaja on välja töötanud.

Tarkvarasüsteemide komponendid

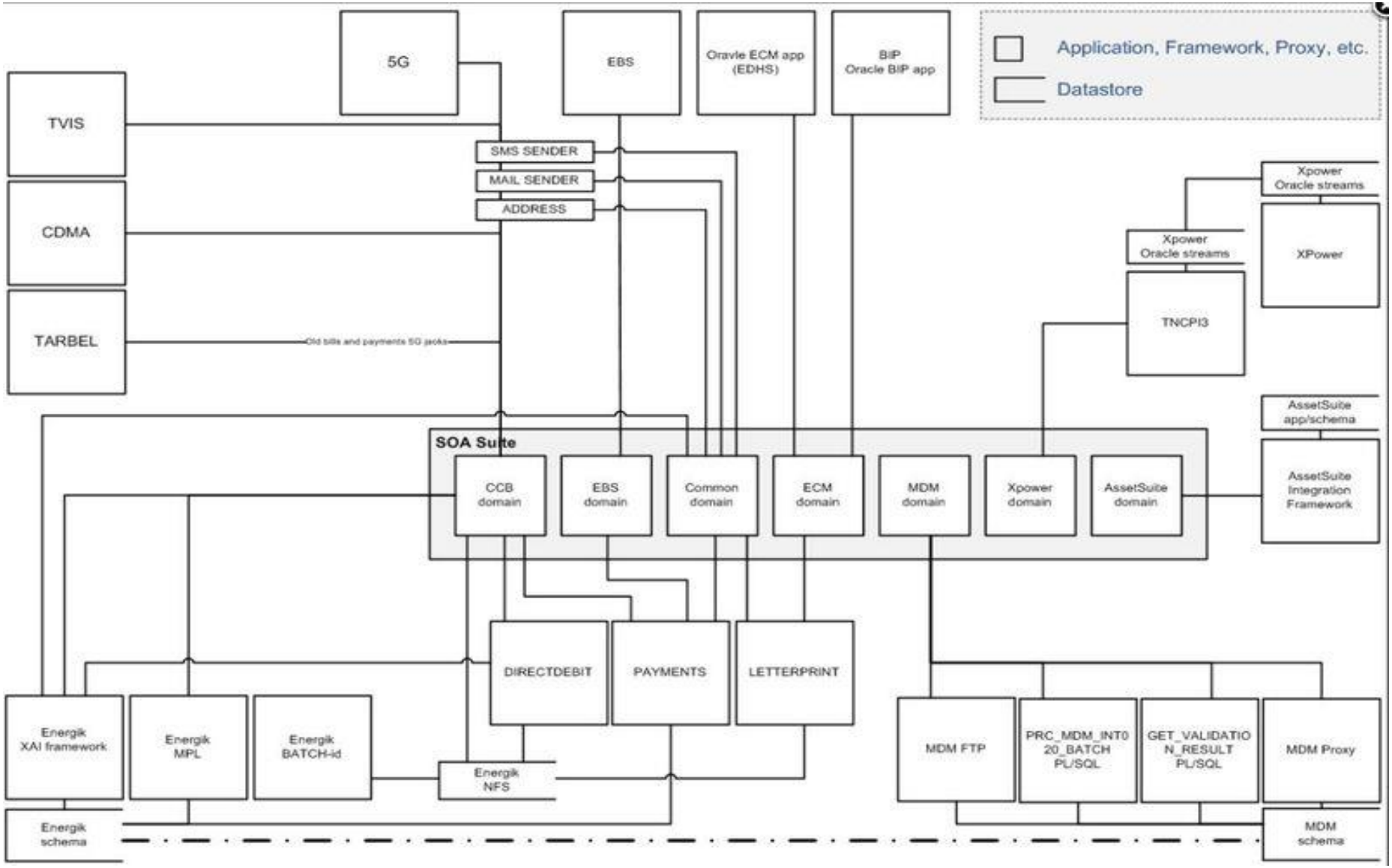
- **Teegid ehk libraries**

Konkreetseid, piiratud funktsioone realiseerivad väikesed komponendid ja nende komplektid. Mõned komplektid on hiigelsuured. Funktsioonide näiteid:

- Trükkimine
- Faili kirjutamine
- Võrguühenduse avamine
- Ringi, joone vms asja joonistamine

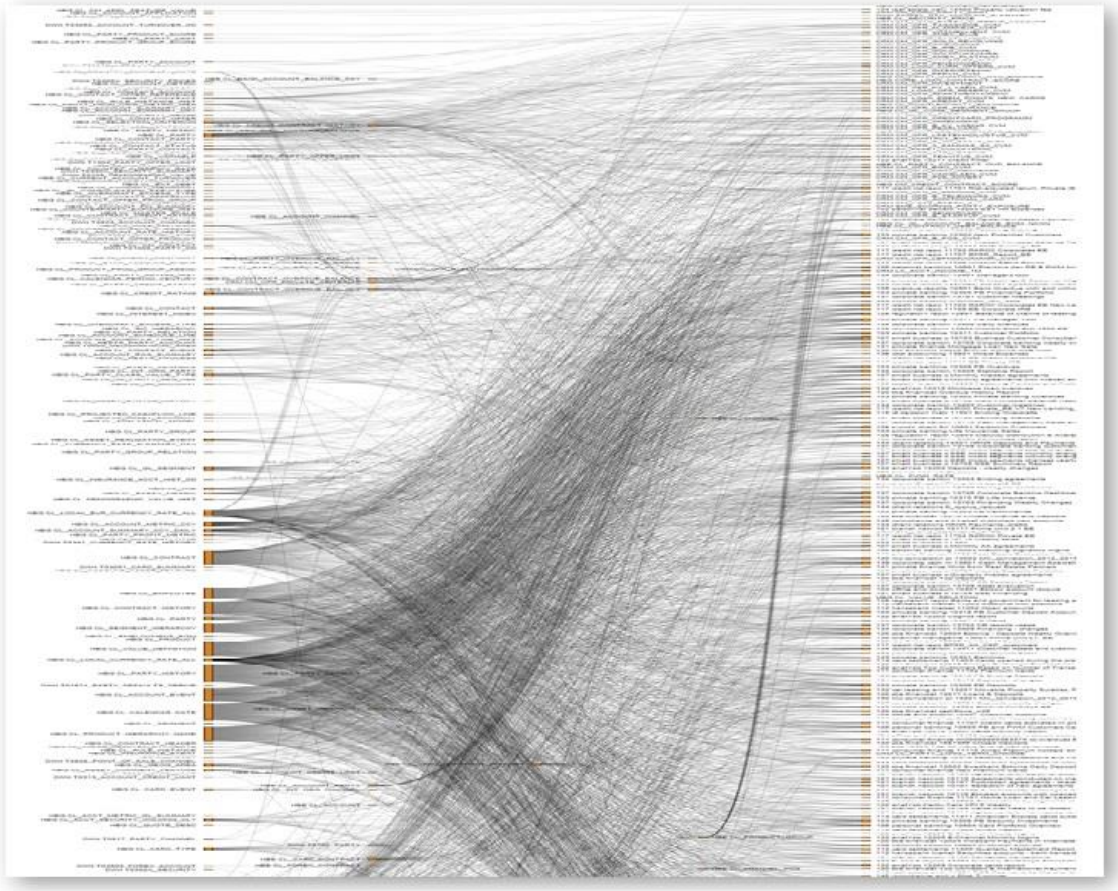
Selliseid komponente levitatakse enamasti komplekti kompilaatorite ja muude tarkvara-arendusvahenditega ning nad on tüüpiliselt kasutatavad ainult selle konkreetse programmeerimiskeele ja arendusvahendi koosseisus.

Üks suur eesti infosüsteem: iga blokk on omaette keerukas süsteem oma andmebaasidega



Andmete liikumise keerulised ahelad

Andmete ja raportite seoste näide



Meeldetuletus: algoritmilised keeled ja kirjelduskeeled

- **Algoritmilised (programeerimiskeeled)**
  - C, C++, Basic, Java, Python, PHP, assembler ....
- **Kirjelduskeeled (spetsifitseerimiskeeled)**
  - HTML, SQL, XML, RDF, TeX,...
- **Praktiline süntees:**
  - algoritmilised keeled manipuleerivad kirjelduskeele abil antavate objektidega
  - näited: Javascript ja HTML., C/Java/Python/... ja SQL, ...

Valdkonniti domineerivad programmeerimiskeeled

- Universaalseim, välja arvatud tippkiirust nõudvad või embedded või brauseri- või operatsioonisüsteemi-rakendused: **Java**
- Maksimaalset kiirust nõudvad rakendused, embedded ja süsteemprogrammeerimine: **C, C++**
- Andmetöötlus ja skriptid ilma kasutajaliideseta: **Python, Java, C, Go, Perl, Ruby**
- Windowsi kasutajaliidesega rakenduste programmeerimine: **C#, VisualBasic, C, (Java)**
- Maci ja iPhone programmeerimine**: Swift, Objective-C**
- Androidi programmeerimine: **Java, (C)**
- Veebibrauseri programmeerimine: **Javascript**
- Veebirakenduse programmeerimine: **PHP, Javascript, Python, Ruby, Java, Go, C , Perl** ..
- Spetsiifilised rakendused: vastavalt vajadusele

Arendusvahendid

Programmi töötamiseks:

- Kompilaator
- Interpretaator

Programmi kirjutamiseks:

- Sobivad tekstiredaktorid
- Visuaalsed arendusvahendid

Suure hulga lähtekoodi halduseks:

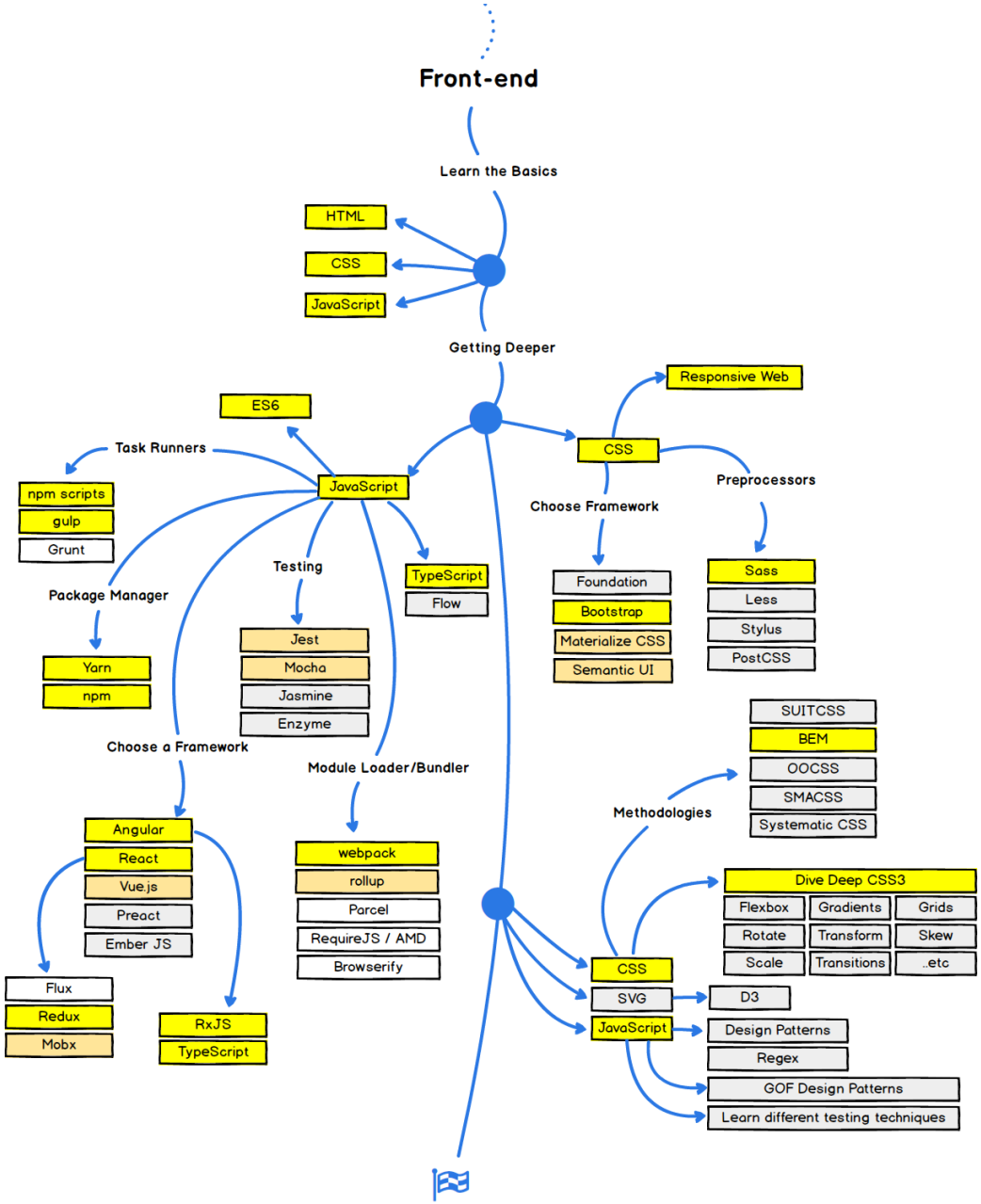
- Versioonikontroll (git, subversion, ...)
- Kompileerimissüsteemid (make, automake, Ant, Gradle, Webpack,...)
- Integratsioonisüsteemid (Jenkins, Travis CL, ...)

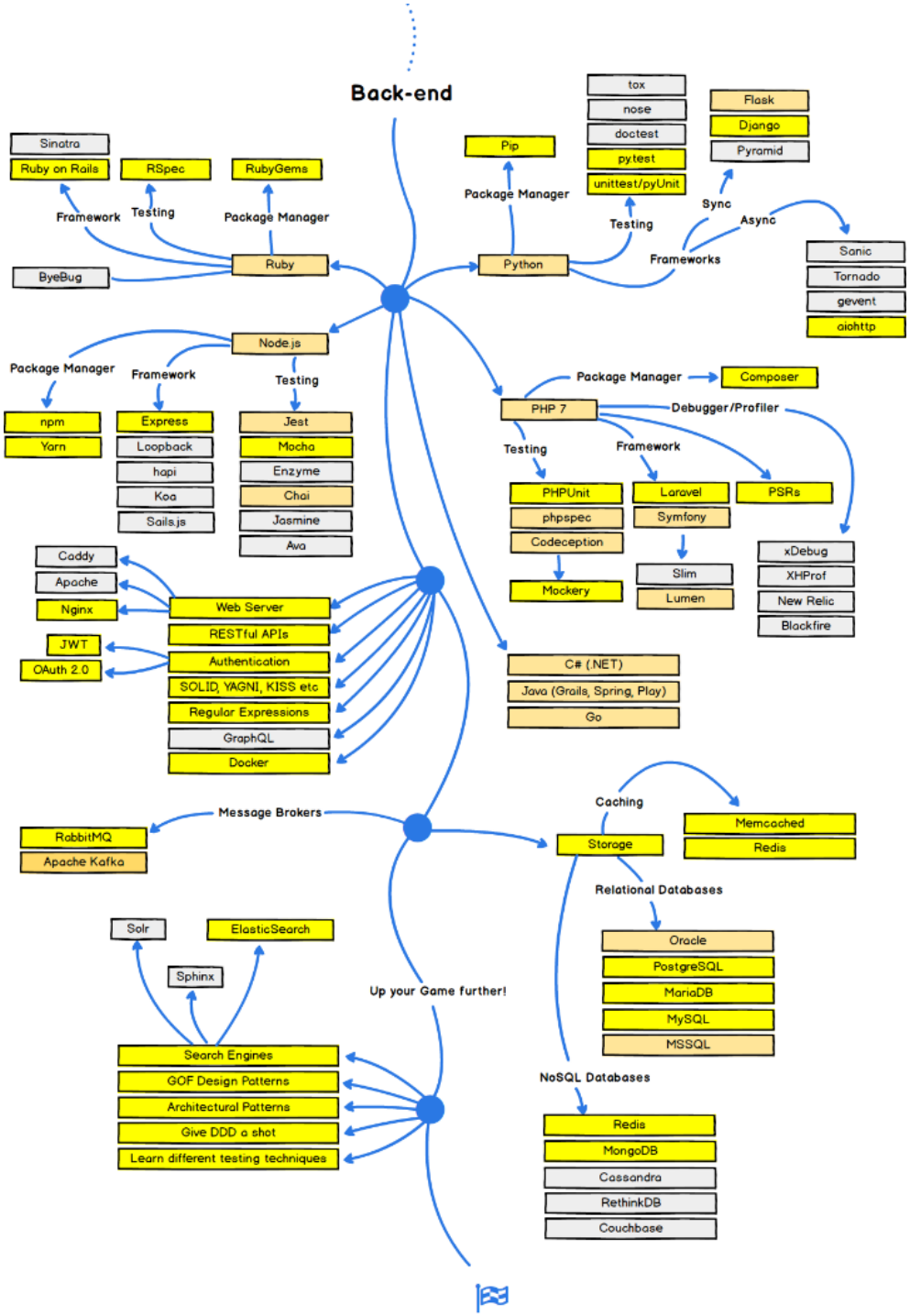
Testimiseks ja monitoorimiseks:

- Testimise abivahendid ja automaattestide süsteemid (Valgrind, Jtest, ...)
- Automaatmonitoorimise süsteemid

Veebitehnoloogiad: näide vahenditest

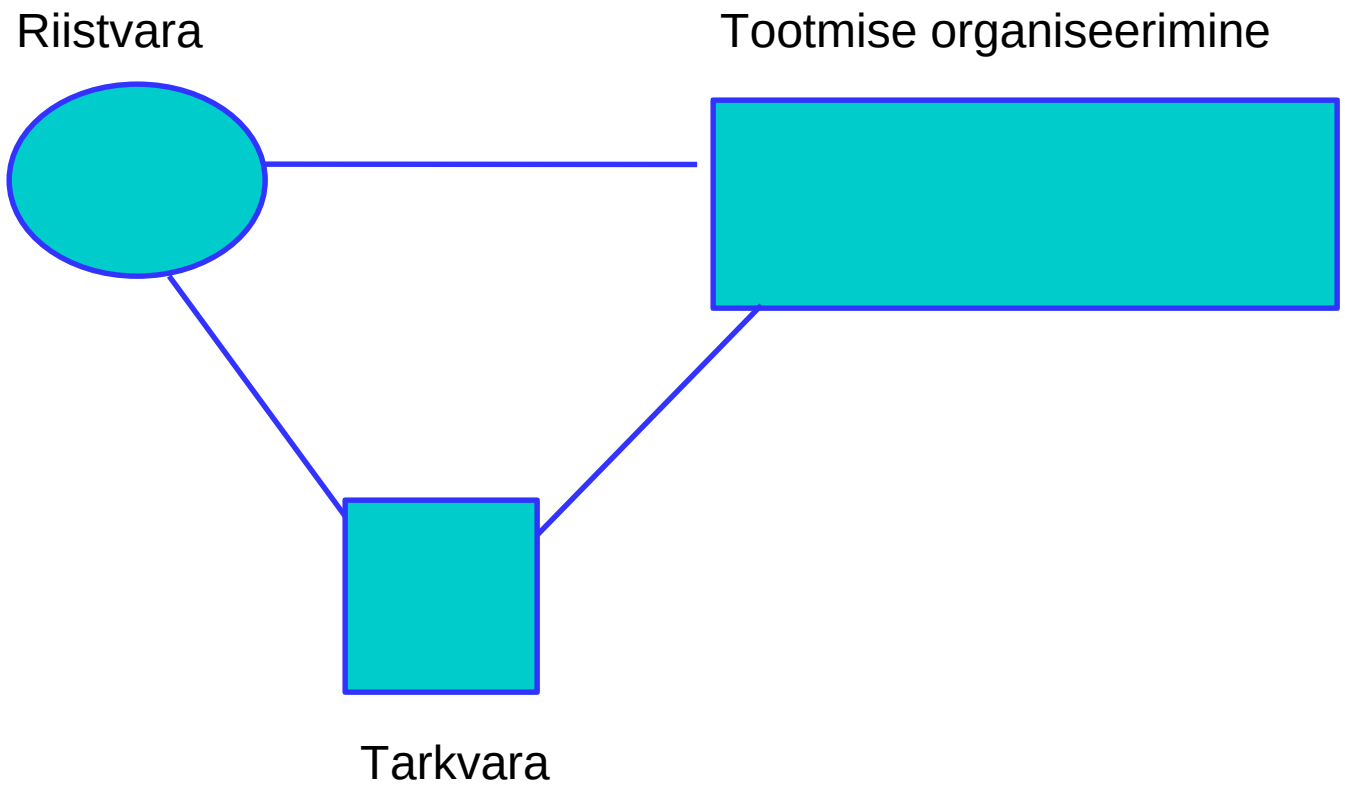
https://codeburst.io/the-2018-web-developer-roadmap-826b1b806e8d


Front-end
Learn the Basics
HTML
CSS
JavaScript
Getting Deeper
ES6
Task Runners
npm scripts
gulp
Grunt
JavaScript
Package Manager
Yarn
npm
Testing
Jest
Mocha
Jasmine
Enzyme
TypeScript
Flow
Choose a Framework
Angular
React
Vue.js
Preact
Ember JS
Flux
Redux
Mobx
RxJS
TypeScript
Module Loader/Bundler
webpack
rollup
Parcel
RequireJS / AMD
Browserify
Responsive Web
CSS
Choose Framework
Foundation
Bootstrap
Materialize CSS
Semantic UI
Preprocessors
Sass
Less
Stylus
PostCSS
Methodologies
SUITCSS
BEM
OOCSS
SMACSS
Systematic CSS
CSS
Dive Deep CSS3
Flexbox
Gradients
Grids
Rotate
Transform
Skew
Scale
Transitions
..etc
SVG
D3
JavaScript
Design Patterns
Regex
GOF Design Patterns
Learn different testing techniques


Back-end
Sinatra
Ruby on Rails
RSpec
RubyGems
Framework
Testing
Package Manager
ByeBug
Ruby
Pip
Package Manager
tox
nose
doctest
pytest
unittest/pyUnit
Testing
Flask
Django
Pyramid
Sync
Async
Frameworks
Python
Sanic
Tornado
gevent
aiohttp
Node.js
Package Manager
Framework
Testing
npm
Yarn
Express
Loopback
hapi
Koa
Sails.js
Jest
Mocha
Enzyme
Chai
Jasmine
Ava
PHP 7
Package Manager
Composer
Debugger/Profiler
Testing
Framework
PHPUnit
phpspec
Codeception
Mockery
Laravel
Symfony
Slim
Lumen
PSRs
xDebug
XHProf
New Relic
Blackfire
Caddy
Apache
Nginx
Web Server
RESTful APIs
JWT
OAuth 2.0
Authentication
SOLID, YAGNI, KISS etc
Regular Expressions
GraphQL
Docker
C# (.NET)
Java (Grails, Spring, Play)
Go
Message Brokers
RabbitMQ
Apache Kafka
Caching
Storage
Memcached
Redis
Relational Databases
Oracle
PostgreSQL
MariaDB
MySQL
MSSQL
Solr
ElasticSearch
Sphinx
Search Engines
GOF Design Patterns
Architectural Patterns
Give DDD a shot
Learn different testing techniques
Up your Game further!
NoSQL Databases
Redis
MongoDB
Cassandra
RethinkDB
Couchbase

IT süsteemide paradigmade areng

Ajalooline vaade kolmele osale:



Pikaajalised arhitektuurifaasid

1945-1970 **1970-1995** **1995-2020?**

A) Riistvara arhitektuurifaasid

Suurarvutid - Mikroarvutid - Võrgusüsteemid

B) Tarkvaraplatvormide arhitektuurifaasid:

assembler, puhtad keeled -
teegid, arendusvahendid, komponendid -
komponentide sidumine

C) Tootmise organiseerimise arhitektuurifaasid:

suurfirma, avatud - väikefirma, suletud -
vabad komponendid, sidumine, hooldus

Priorities for software development

Three main consumers of time and effort:

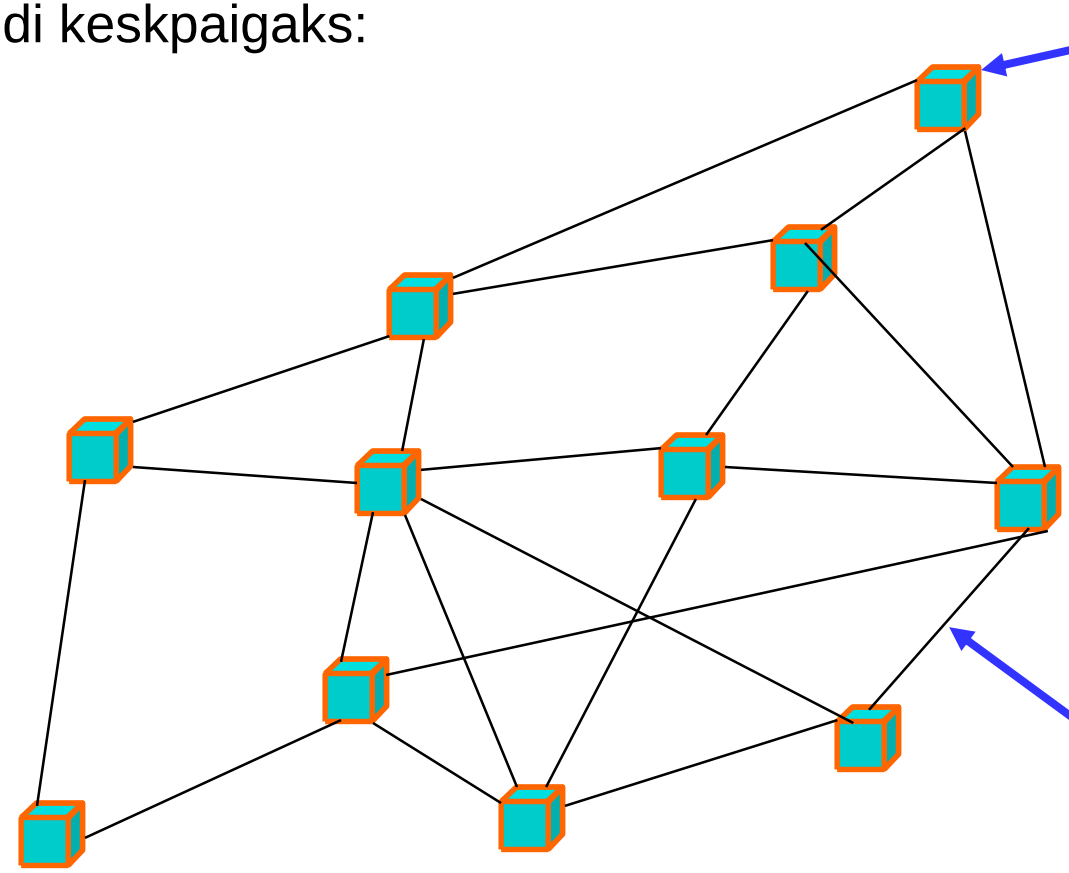
- Understanding the business processes and needs.
- Understanding the exact contents of existing data.
- Writing code.

The second component - understanding existing data -
is growing and will keep growing for foreseeable future.

Why?

Intelligentne võrk!?

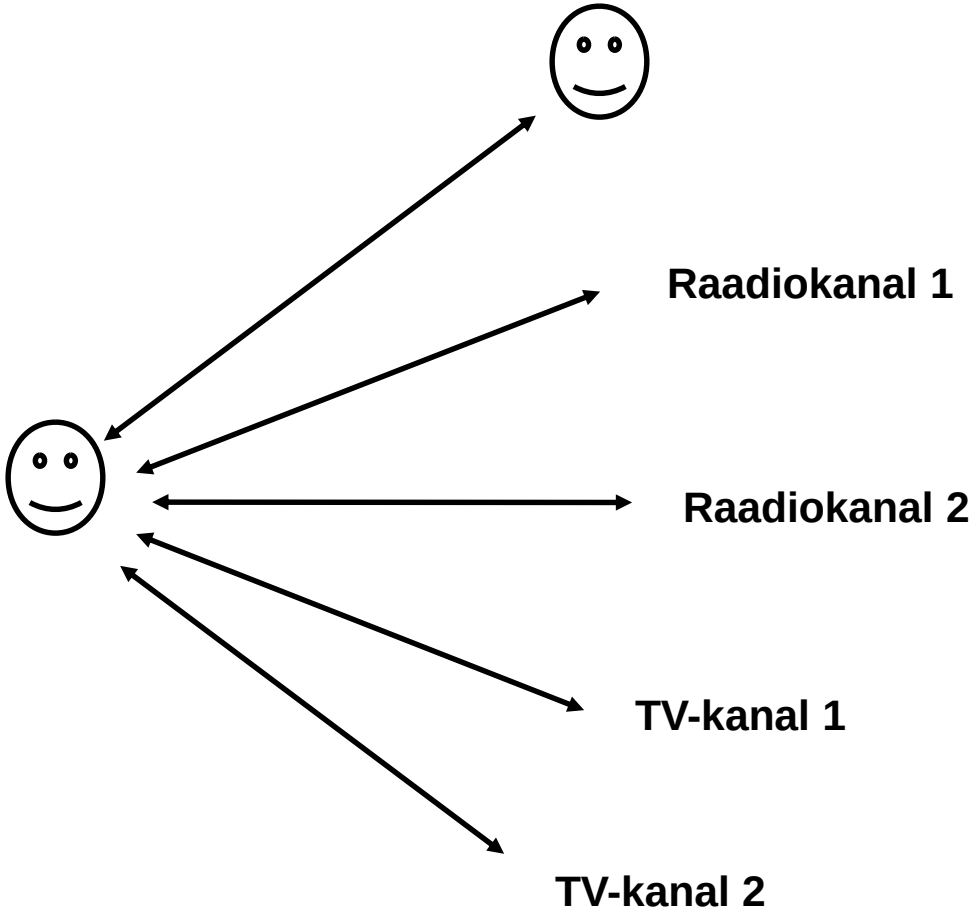
Sajandi keskpaigaks:



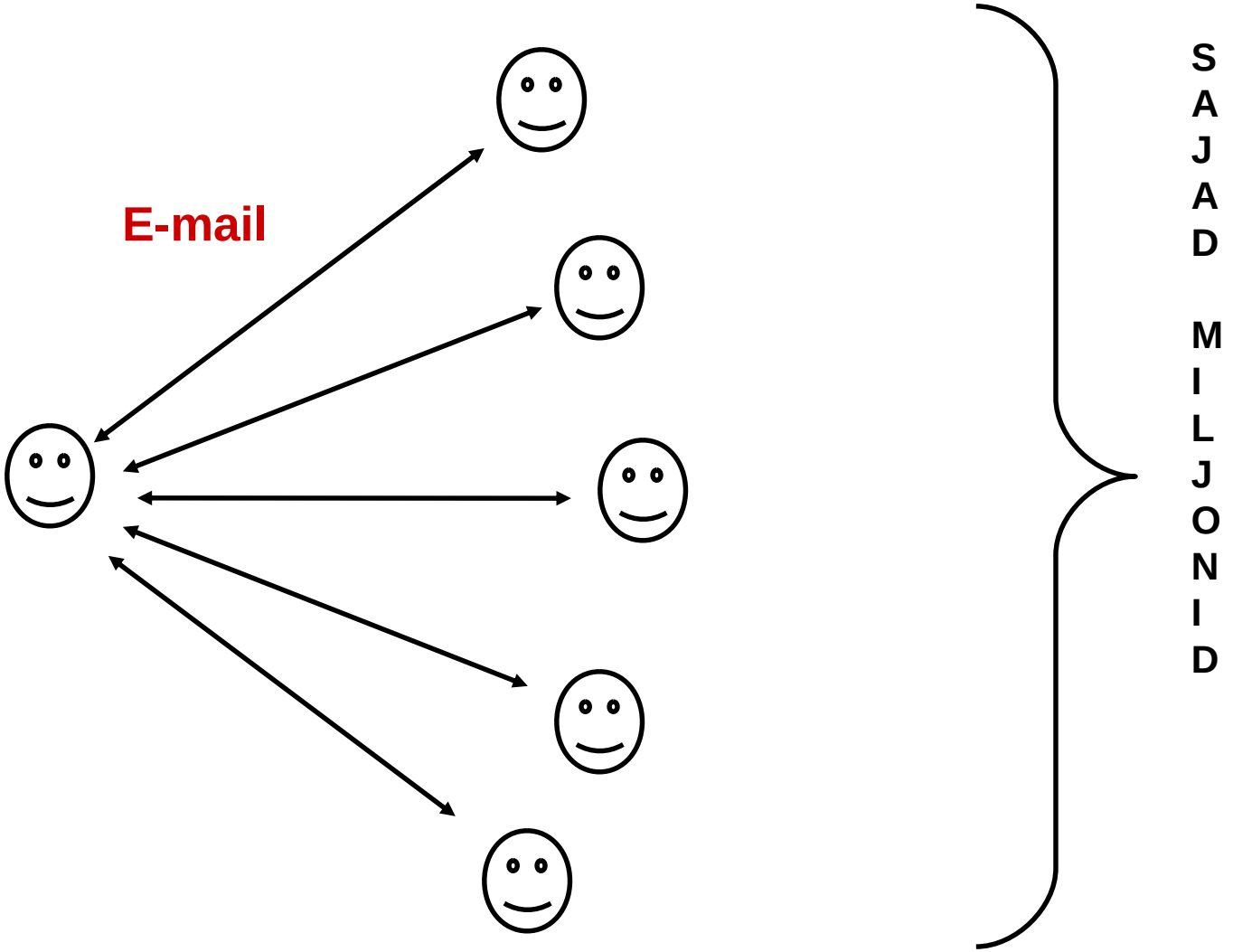
Iga arvuti on praegustest tuhandeid kordi võimsam

Võrgus on praegusest tuhandeid kordi rohkem infot ja teenuseid

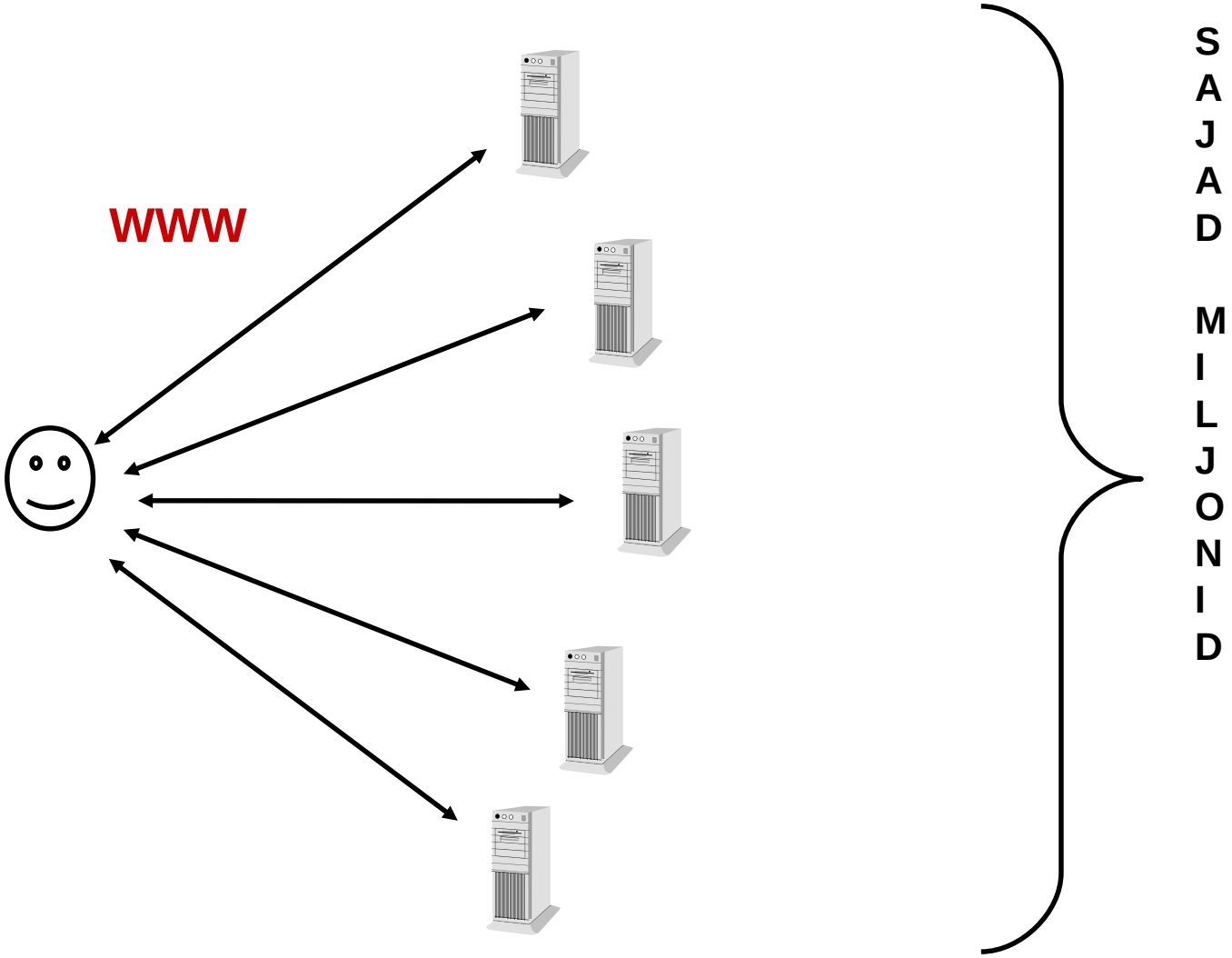
Internet 0:
inimene <--> veidi inimesi, raadio, TV



Internet 1: inimene <--> hulga inimesi

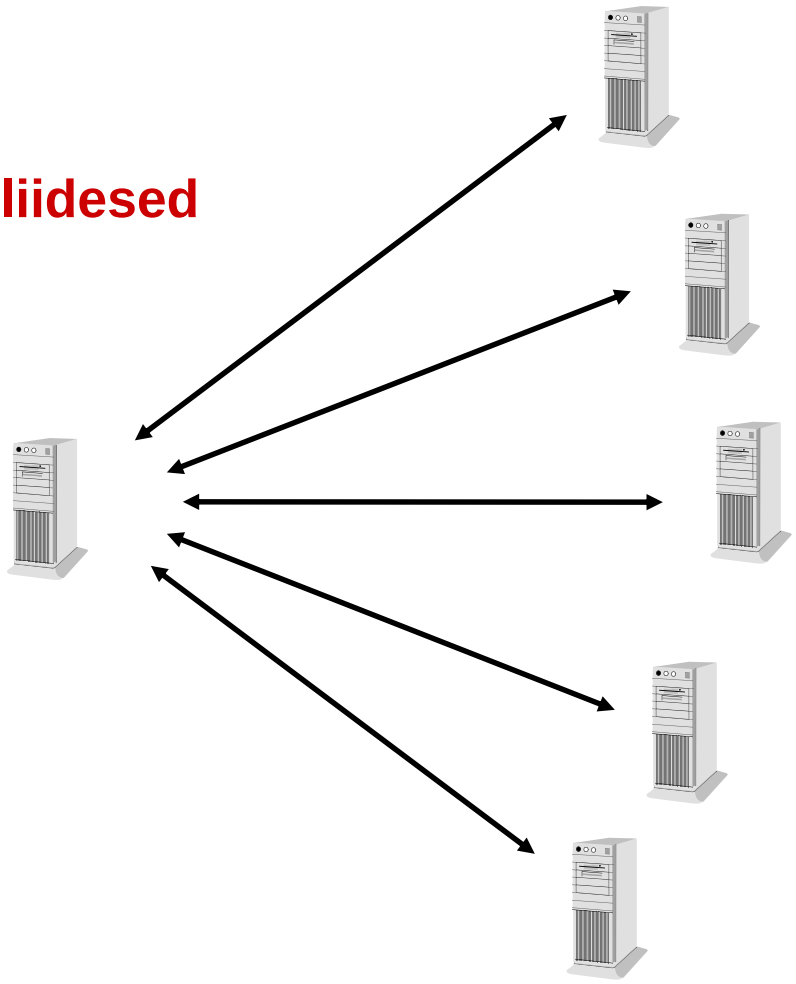


Internet 2: inimene <--> hulga masinaid



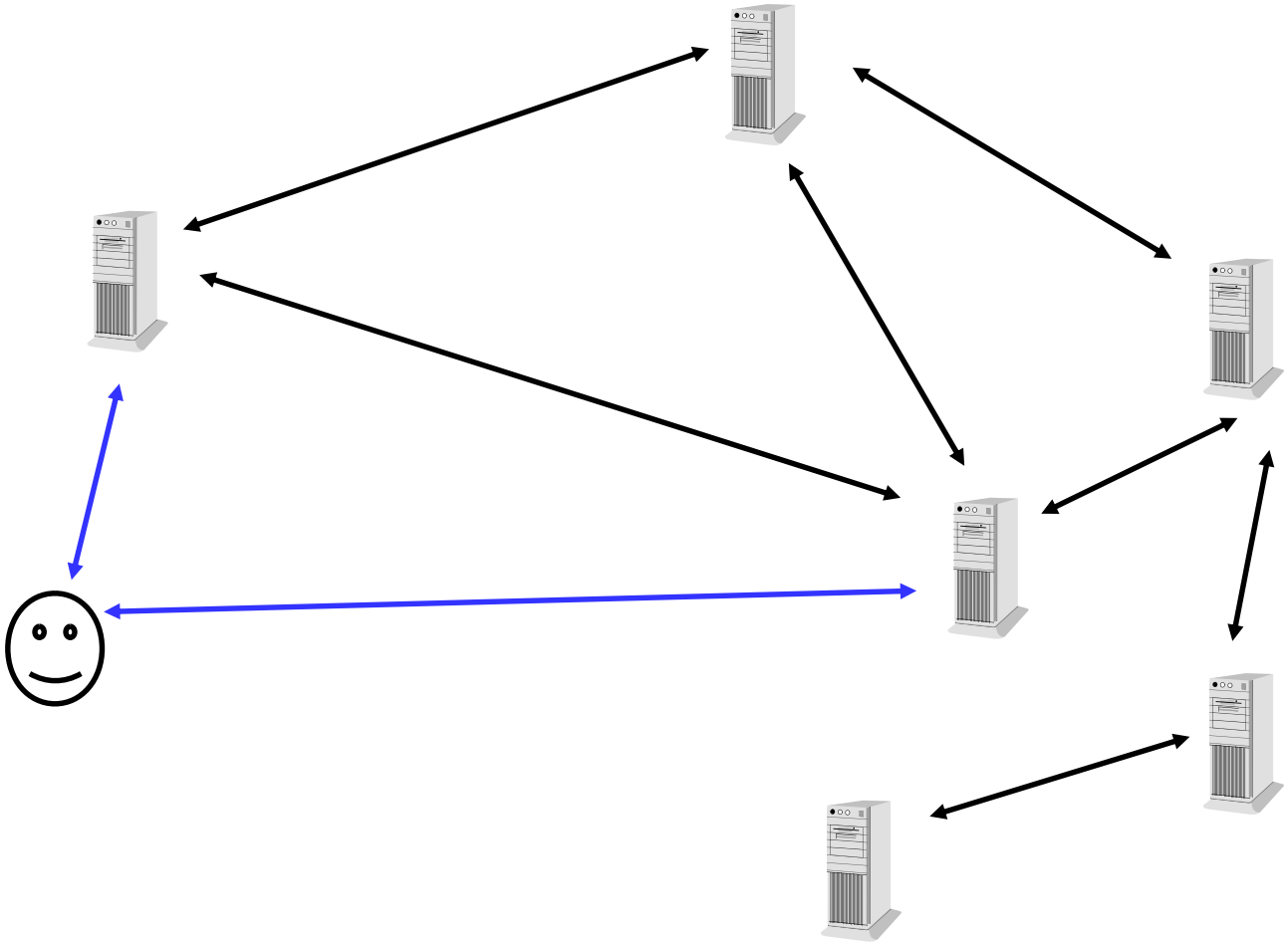
Internet 3: masinad <--> hulga masinaid

Automaatliidesed



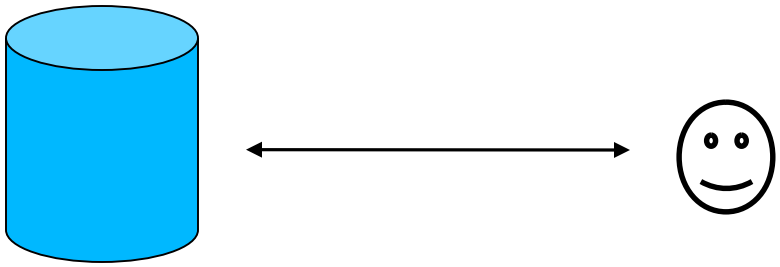
SAJAD MILJONID

Internet 3: inimene ??



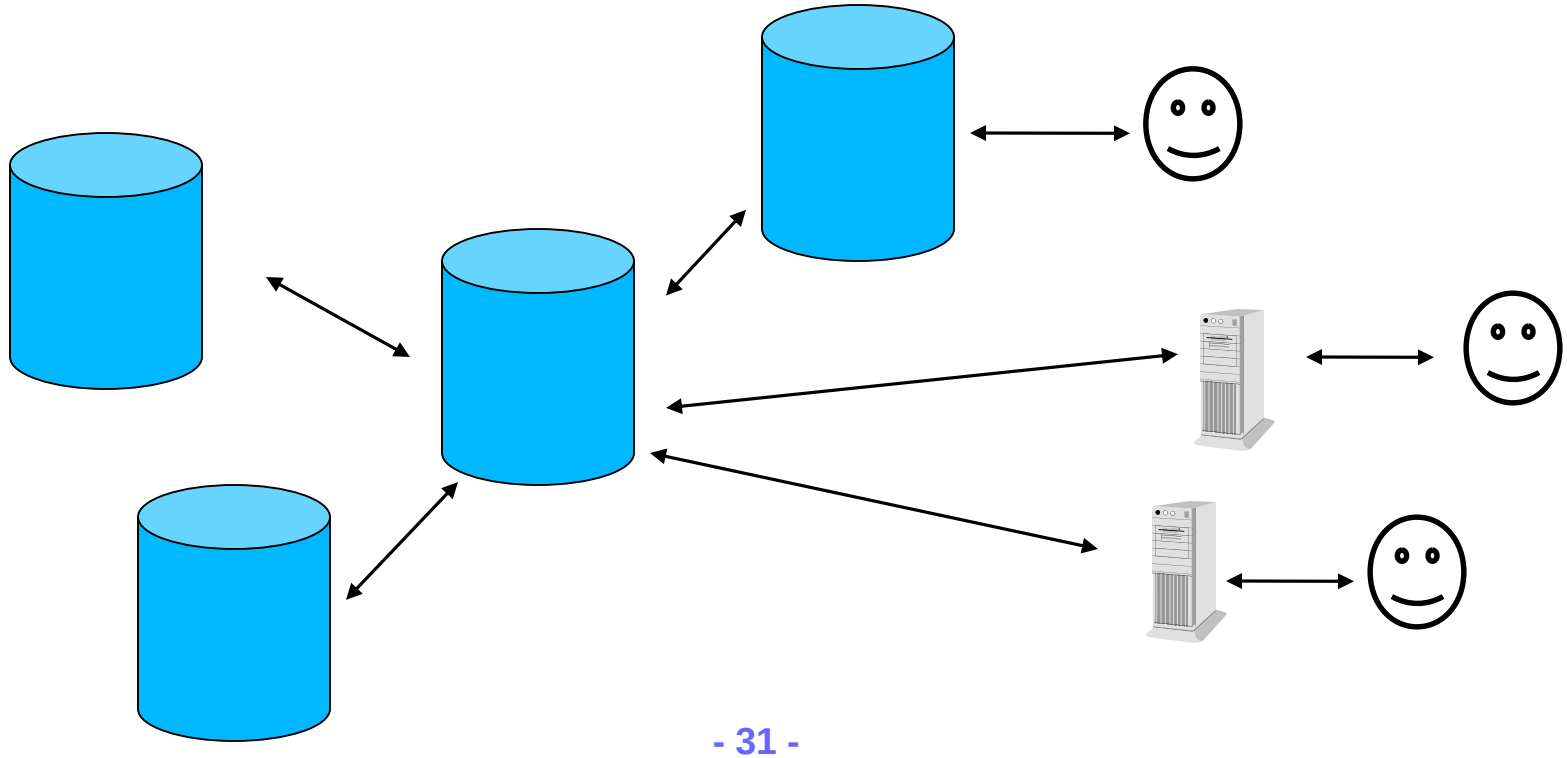
Architecture I

Compact standalone software



Architecture II

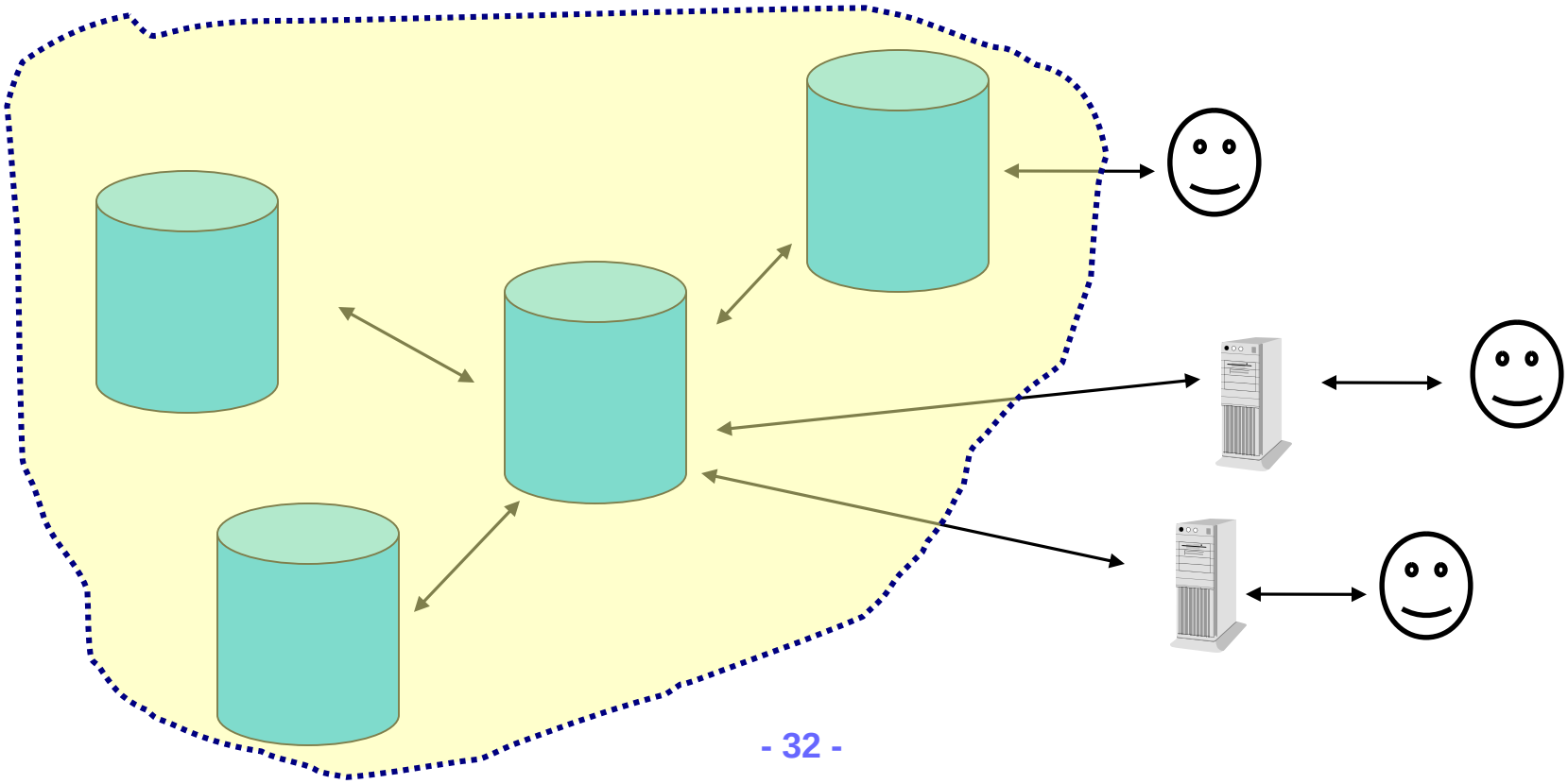
Classical client-server applications



Architecture II : similarities to I

All applications controlled by the developer:

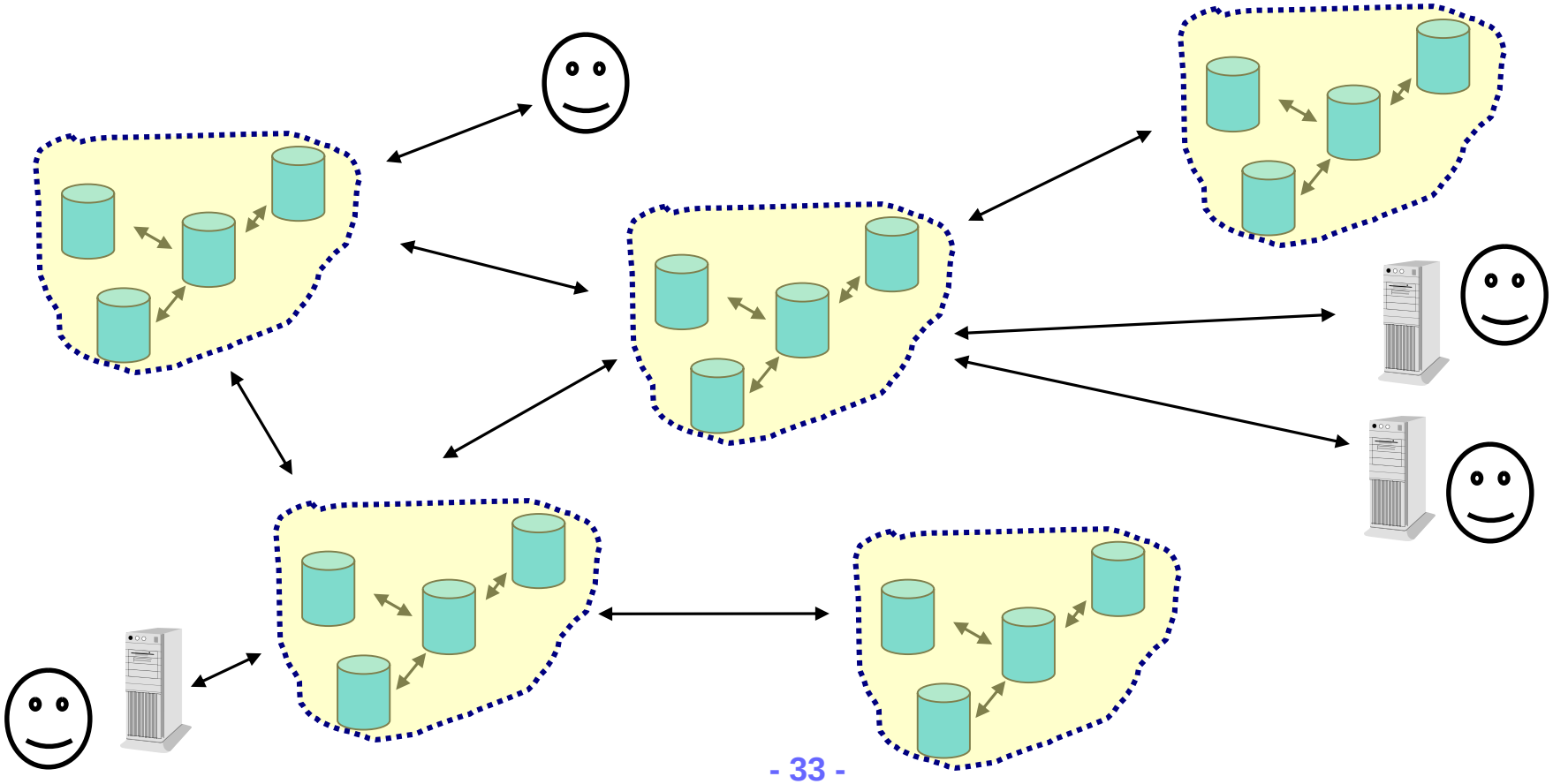
- located in one machine or
- distributed in the machines of the company/client



Architecture III

Applications **not** controlled by the developer:

- you only have access to them
- system functions like a society of applications



Architecture III: some examples

- Apps based on web services
- Global comparison/search systems
- Estonian public sector: xtee complex queries
- Document management between different organisations
- Large financial apps: banks etc
- Sales systems
- Military apps
- ...

Architecture III: kill chain example

- Context: Yugoslavia, Iraq, Afganistan ...

- Warfighting integration.
- We're basically tasked to close the
- seams in the kill chain.
- The kill chain is a euphemism for the process by which we

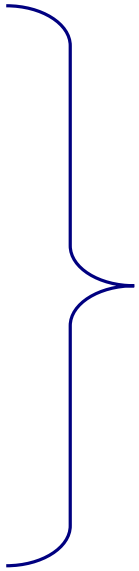
  - identify targets
  - find them
  - fix it
  - track it
  - target
  - execute
  - check results

Architecture III: kill chain example

- Context: Yugoslavia, Iraq, Afganistan ...

- Warfighting integration. We're basically tasked to close the seams in the kill chain. The kill chain is a euphemism for the process by which we

  - identify targets
  - find them
  - fix it
  - track it
  - target
  - execute
  - check results



Every link is a separate country with its own information systems

Architecture III: kill chain example

- Context: Yugoslavia, Iraq, Afganistan ...

- Warfighting integration. We're basically tasked to close the seams in the kill chain. The kill chain is a euphemism for the process by which we

  - identify targets
  - find them
  - fix it
  - track it
  - target
  - execute
  - check results

Every link is a separate country with its own information systems

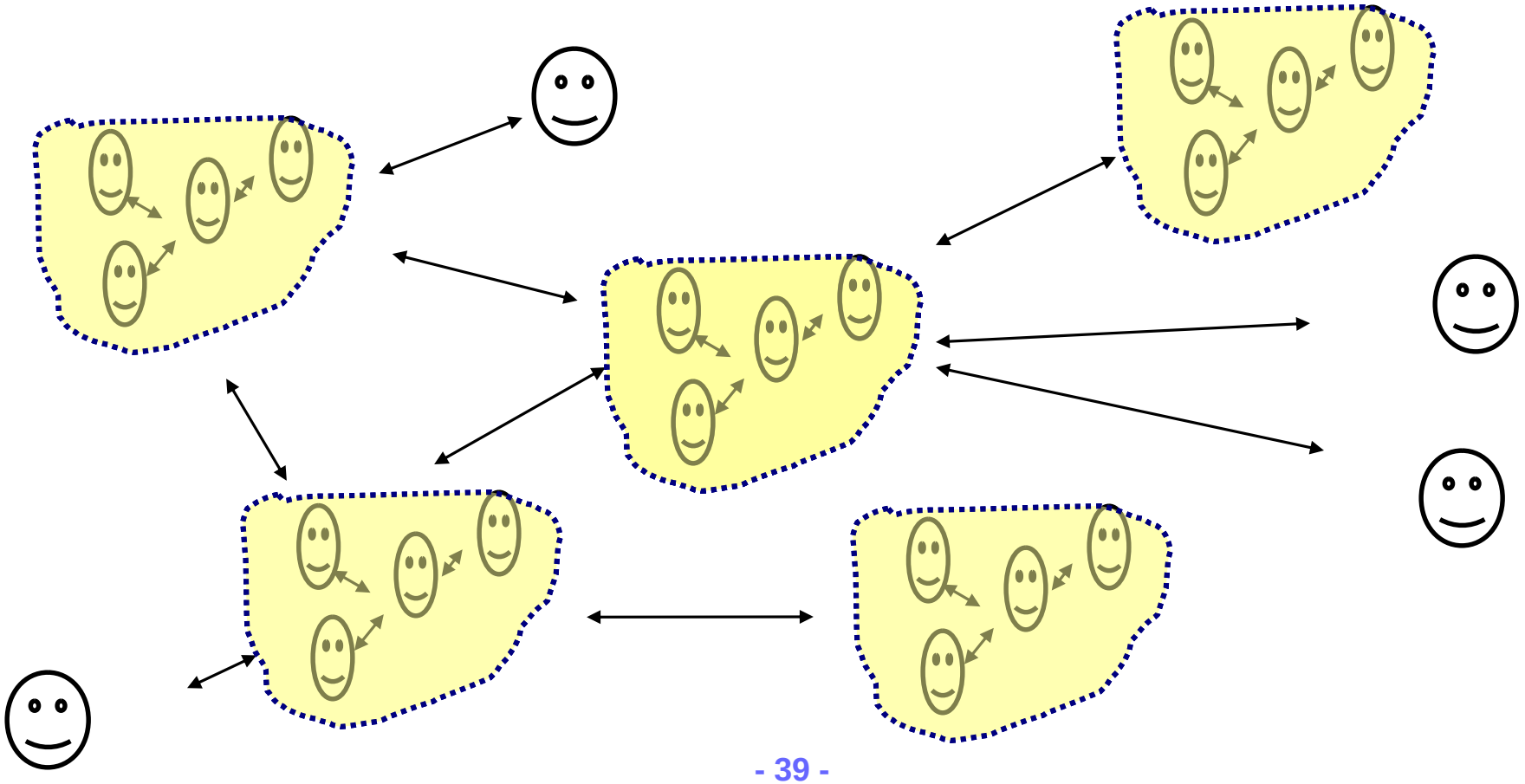
Countries in the chain are often exchanged or replaced!

Main problems for the developer

- **Get access** to data from the foreign system
- **Understand** what does this data exactly mean
- **Convert** foreign data to data structures of our own app and vice versa

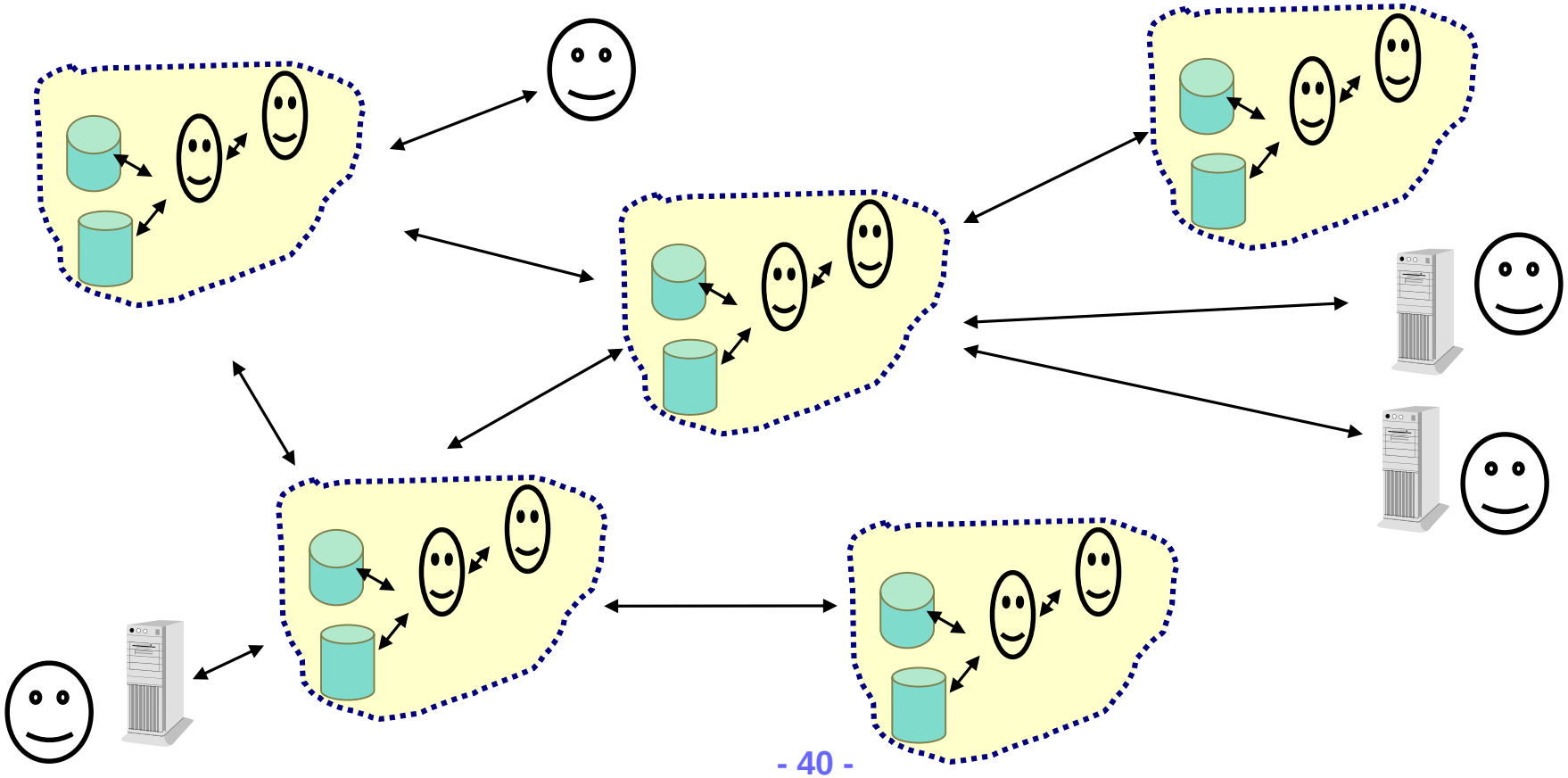
Getting access and understanding?

Social networking. Connected groups of people:



Architecture III revisited

Social groups consist of both people and apps



Philosophy. Society and people.

- We could think that do have AI already in some sense:
  society is a large animal
  with an intelligence of its own.

- **People - society** is similar to
  cells - animal

Tarkvara litsentsid

- Litsents on lihtsalt ühepoolne „võta-või-jäta“ leping, et kuidas tarkvara tohib kasutada.
- Kommertslitsentsid on väga mitmekesised: igaüks kirjutab endale sobiva.
- Vabavara on vahel dual-licenced: valid, kas kasutad piirangutega vabavarana või teistsuguste piirangutega kommertslitsentsiga.
- Vabavaralitsentside hulgas on mitmed rohkem/vähem vabad kategooriad: vaata sissejuhatuseks

Vaata ka:

https://www.gnu.org/licenses/license-list.html

Vabatarkvara

- **Mis on vabatarkvara:**
  - tasuta?
  - lahtise koodiga?
  - edasimüügiõigusega?
  - copyright?

- **Ajalugu, eesmärgid, perspektiivid**
  - mis on olnud?
  - mis on tulemas?
  - äriidee?
  - miks vabavara tehakse?
  - kus on vabavara mõistlik kasutada?
  - mida vabavara on endaga kaasa toonud?

GNU ideoloogia:

- vabadus: primaarne on tarkvara vabadus, sekundaarne tasuta kättesaadavus (free as in free speech, not as in free beer)
- ausus: ausam on kasutada vabavara kui piraatkopeerida
- teadmiste vabadus: teadmised, tarkvara on loomu poolest vaba - teadmiste ja tarkvara kopeerimine laiendab ühiskonna majanduslikku võimsust, kaotajaid (rumalamaks jääjaid) pole
- raha saab teenida ka tarkvara
  - toetades ja installeerides ja levitades
  - tellitud täiustusi ja modifikatsioone ehitades
  - ehitades sellist vabavara, mida klient soovib
- motivatsioon senistes kogustes tarkvara luua väheneb?
  - vastuargument: programmeerimine on põnev
  - vastuargument: ehk polegi meil nii palju kallilt makstud programmiste vaja?
  - vastuargument: vabanenud programmeerijate kaader saab teha uusi, senisest keerulisemat softi, mille jaoks seni aega ega ressurssi ei jätkunud

GPL ja litsentsid

- Vaatame eraldi GPL, LGPL ja muid vabavara-litsentse